Week 5 Lab

For an example of how to do the exercises you can refer to my week 5 GitHub repo which is at:

https://github.com/oit-gaden/Web-Development-2019-Winter/tree/master/Examples/Week5

Don't forget to push your code to GitHub in a folder called week5 and your Docker images (you will have 2) to Docker Hub.

Exercise 1 - Install .NET Core SDK

1. Install .NET Core **SDK** 2.2 from https://dotnet.microsoft.com/download

2. Run **dotnet --version** to make sure you have the dotnet CLI installed. You may need to close and reopen a command/shell window first.

3. Run **dotnet new --help** to see all of the types of things you can create from the CLI.

Exercise 2 - Create the sample .NET Core WebApi (REST Service)

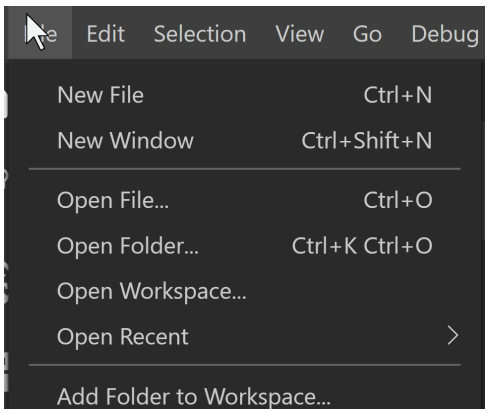
1. Make a copy of your week4 folder and name it week5. This will take a bit of time as the node_module folder contents i quite large.

2. Change folder to week5 and create the folder webapi

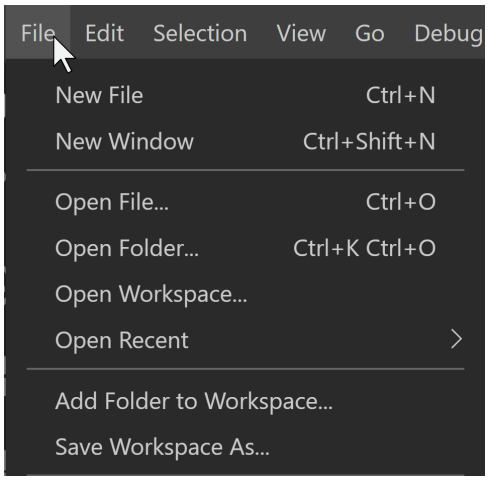
3. Change to the folder week5/webapi and create the sample WebApi/REST Service by running the command: **dotnet new webapi**

Exercise 3 - Create a VSCode workspace to contain both the webapp and the webapi projects. (Optional if you are not using VSCode)

1. Start up VSCode and add the two folders to the workspace by using the menu item "Add Folder to Workspace ..." shown below:

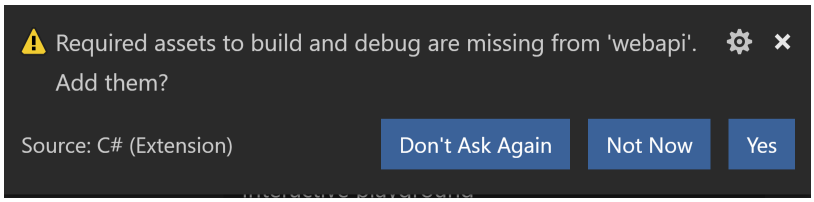


2. Save the workspace in the week5 folder using the menu item "Save Workspace As ..." show below:



This will allow you to open both projects in VSCode by just opening the workspace file you just created.

3. If you see the following dialog answer yes.



Exercise 4 - Run the sample WebAPI/REST Service

1. Install C# Extension for VS Code

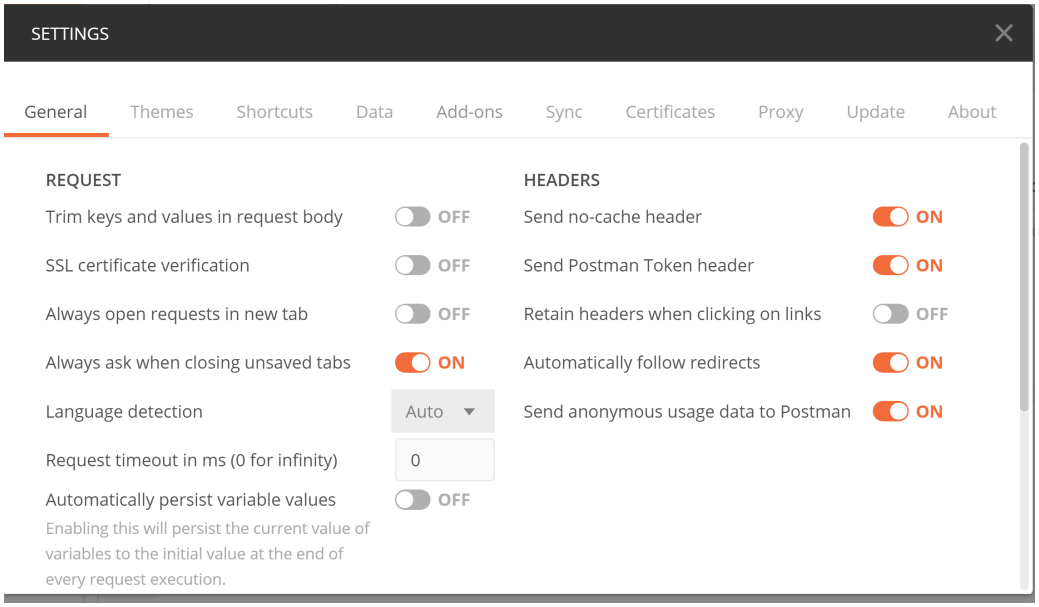
   Install from https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp

2. Select **webapi** from the VSCode Explorer and then **"Start Without Debugging"** from the **"Debug"** menu. This will display

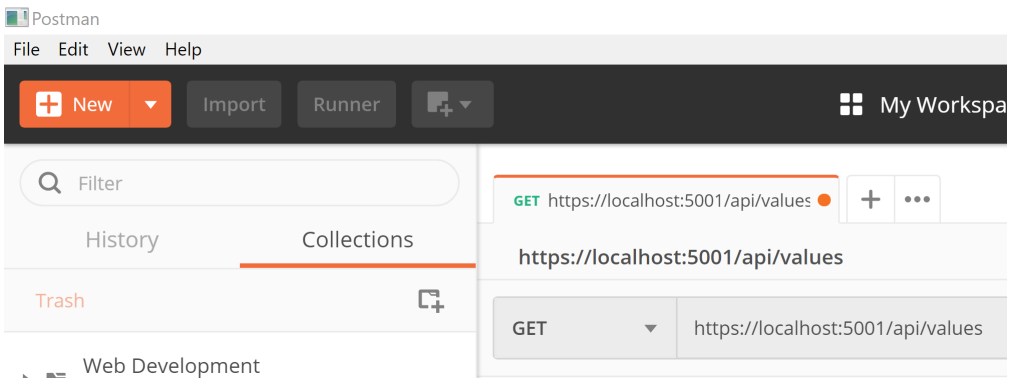
a list of runtime in a small menu.
Select .NET Core. This will then create a launch.json file to enable the launching of the application. This only needs to be done once. The webapi should then
start. If not try selecting **"Start Without Debugging"** again. Once started successfully, a browser window should then appear. We'll access the API in the next
step using Postman.

3. Install Postman from https://www.getpostman.com/ if you haven't already installed it.

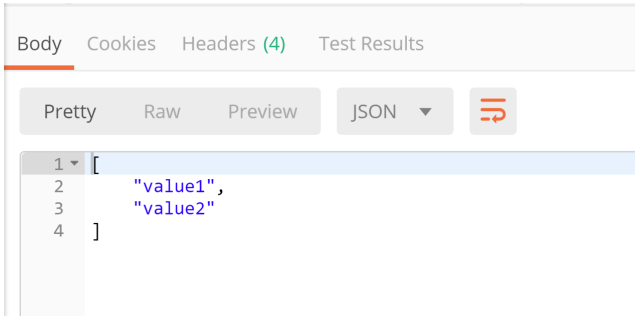
4. Configure Postman to turn off SSL certificate verification. If you don't do this Postman will display a security error. You get to the screen below by clicking on the
wrench in the top right corner of Postman. Then make sure "SSL certificate verification" is OFF.



5. In Postman enter the URL shown below:



6. You should see the return values in the bottom part of Postman as below:



7. FInd the ValuesController in the webapi project code where these values are being generated.

Exercise 5 - Create the controllers for your API/REST service

1. In the **webapi** project, create a controller like the **ValuesController** for both students and persons. You only need to include the "get" method for now.
   Don'f forget to change the class names accordingly. You can remove the **ValuesController** if you wish.

2. Replace the Startup.cs file with the one from my GitHub repository. I made some needed changes to the default one created. I'm not going to go into detail
   what the changes were for but ask me if you would like to know.

3. Look at the example ProductController in my GitHub repository and follow the same pattern for returning persons and students from your two controllers.

4. Create a model for both student and person like I did for product.

5. Create an "in memory" database like the one in my example to "store" your student and person data. Look at the folder "week5/webapi/Database" in my example.

6. Try accessing your API from Postman to retrieve both students and persons much like you did in exercise 4 for the values in the sample API.

Exercise 6 - Add call to your API/REST service from your Web app

1. In a command window/shell go to the **webapp** folder and run the commands **npm install vue-axios and npm install axios**. These are the libraries we are using for issuing HTTP calls from the
   Web app.

2. Add logic to the Vue components for both persons and students to retrieve the data from your REST service rather than from a hardcoded JSON string. See my
   Product.vue component for how to do this.

3. You'll need to add Axios and VueAxios to the main.js file under the src folder as follows:

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
import axios from 'axios'
import VueAxios from 'vue-axios'
import App from './App'
import router from './router'
```

   You'll also add the line: **Vue.use(VueAxios, axios).** See my main.js to see where to put it.

Exercise 7 - Build and run your Web app and api in Docker containers

1. Copy the Dockerfile from my GitHub repository webapi folder to your webapi folder.

2. Copy the docker-compose.yml file from my GitHub week5 example folder to your week5 folder. Remove the webapi-node section.

3. You must build your webapp first by running the command: **npm run build.**

4. Run the command **docker-compose up -d --build** from your week5 folder. You should see the images being built and the containers started.
   After the command completes runthe command **docker ps** to see the running containers **webapp** and **webapi**.

5. Try accessing your webapp at http://localhost:8080/ You should be able to go to the students and persons pages and see the tables populated

from data coming from your API.

6. To stop the containers run the command **docker-compose down** in the week5 folder. When it completes run **docker ps** and you should now not see your containers running.

7. Push your Docker images to Docker Hub and push your code to GitHub.

Extra Credit - Incorporate the Weather service I used or another one of our choice fromhttp://www.programmableweb.com/